PASSED

lab3Group Computation Statistics

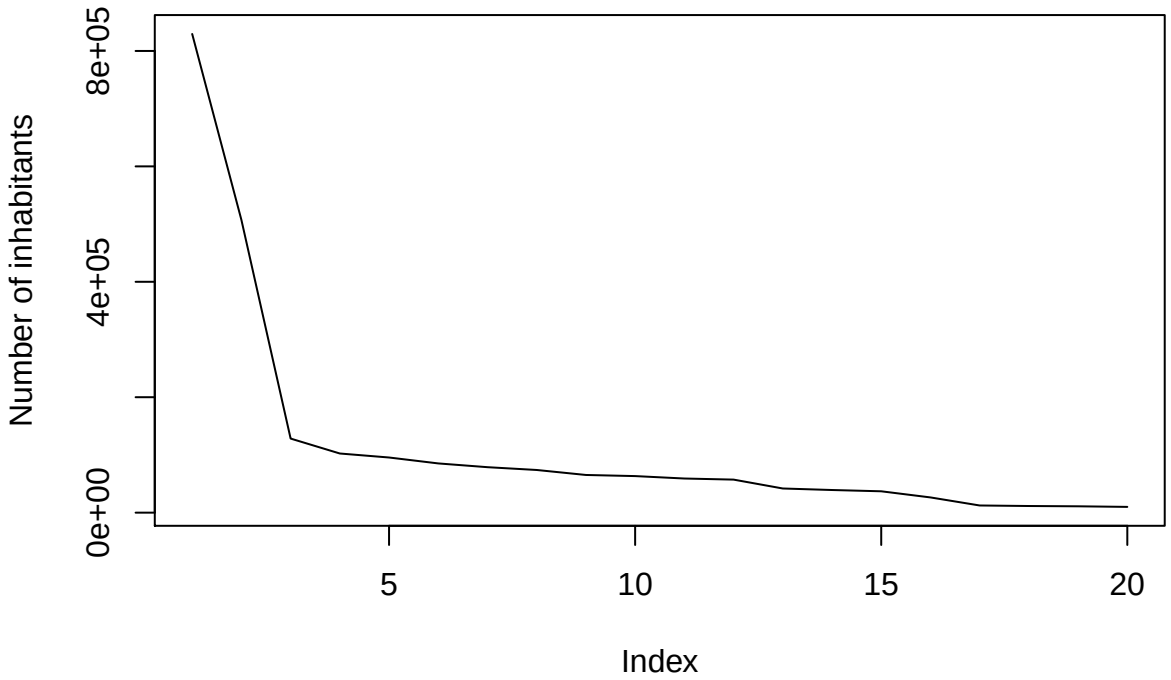
Niclas Lovsjö,Maxime Bonneau

19 februari 2016

Assignment 1

4.

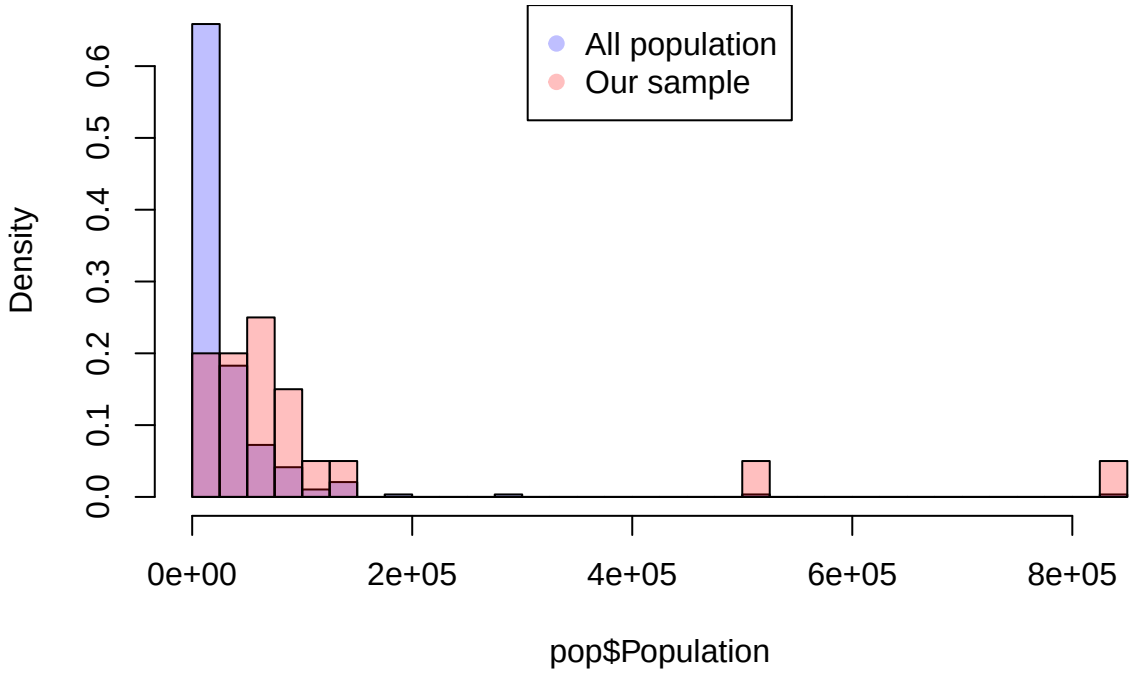
When running this program, the following cities are returned: Sundsvall, Vetlanda, Södertälje, Borås, Stockholm, Gotland, Kungsbacka, Helsingborg, Åtvidaberg, Surahammar, Motala, Kristianstad, Östersund, Sandviken, Åmål, Vaxholm, Sigtuna, Järfälla, Göteborg, Sollentuna



Just above is the ordered size of the cities picked by the program. We can notice that Stockholm has been picked. Since it is the largest city it has higher chances to be chosen. Almost two thirds of the cities have more than 80 000 inhabitants (6 out of 20 exactly). It is a bigger proportion than for all the cities, which has 23 out of 290 cities with more than 80 000 inhabitants.

5.

Following is the histogram of the repartition of cities by size (in blue). In red is the distribution of the cities which were picked up by the program. The distribution is globally the same, but we can notice that we have more big cities than the original distribution.



So we can conclude that our sampling method with the probabilities based on the size of the cities is not totally fair, but it was also the goal of giving different weights to cities, to avoid to pick only small towns, which would have become not representative of the whole population.

Assignment 2: Different distributions

Inverse CDF method

In this assignment we are concerned with the double exponential distribution(DE). We want to come up with a way of generating the DE using a random number $U \sim U(0,1)$.

One simple and immediate way is to note that this is indeed a *double exponential distribution*, which means that we could generate $Y \sim exp(\lambda)$ from $U = F(y)$ with the inverse CDF-method. As we have seen in the textbook(Gentle) and in the lecture(Sysoev), this yields $y = F^{-1}(U) = -log(1-U) = -log(U)$. Then we use another uniform random number, $U1$ say, and "flip a coin" whether this particular generated number should go to the left-hand-support or the right-hand-support, i.e. generate y, then if U1<0.5 do y=-y. We note that this is possible since the distribution is symmetric around its *location*, 0 in this case.

Another way of doing this is to use the $DE(0,1)$ and the inverse CDF-method directly. We first need to find the CDF. We have,

$$F(y) = \int_{-\infty}^{y} f(u)du = \int_{-\infty}^{y} \frac{1}{2}\exp(-|u|)du$$

Since we have the absolute value we have to split into two cases. For $y \leq 0$ we have,

$$\int_{-\infty}^{y} \frac{1}{2}\exp(u)du = \frac{1}{2}exp(y) \tag{1}$$

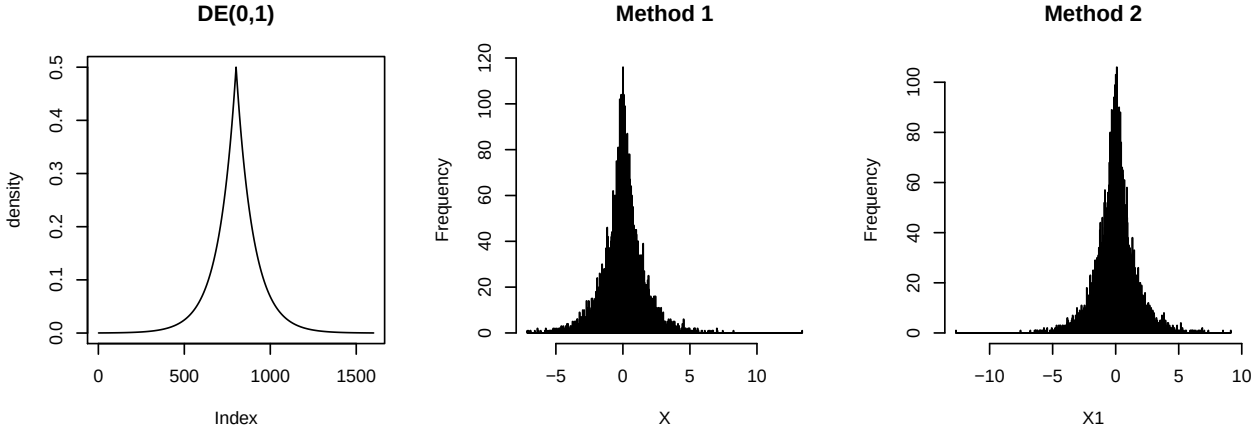
and for $y > 0$,

$$\int_{-\infty}^{0} \frac{1}{2}\exp(u)du + \int_{0}^{y} \frac{1}{2}\exp(-u)du = 1 - \frac{1}{2}exp(-y) \tag{2}$$

Taking the inverse for (1) and (2) separately yields that $y_{(1)} = log(2U)$ and $y_{(2)} = -log(2(1-U))$ respectively. Now inspecting the ranges of (1) and (2) for their respective domains, shows that when $y \in (-\infty, 0)$ then $F(y) \in (0, 1/2)$ and when $y \in (0, \infty)$ then $F(y) \in (1/2, 1)$. So we generate $U \sim U(0,1)$ and whenever $U \leq 1/2$ we use that U with the inverse of case 1, and whenever $U > 1/2$ we use that U with the inverse of case 2.

Another path to go from here, would be to mix (1) and (2) and use the $sign()$-function, which then takes care of the "coinflip" automatically.

Here is a plot that shows that the results are equal, and that the generated distribution is indeed $DE(0,1)$:

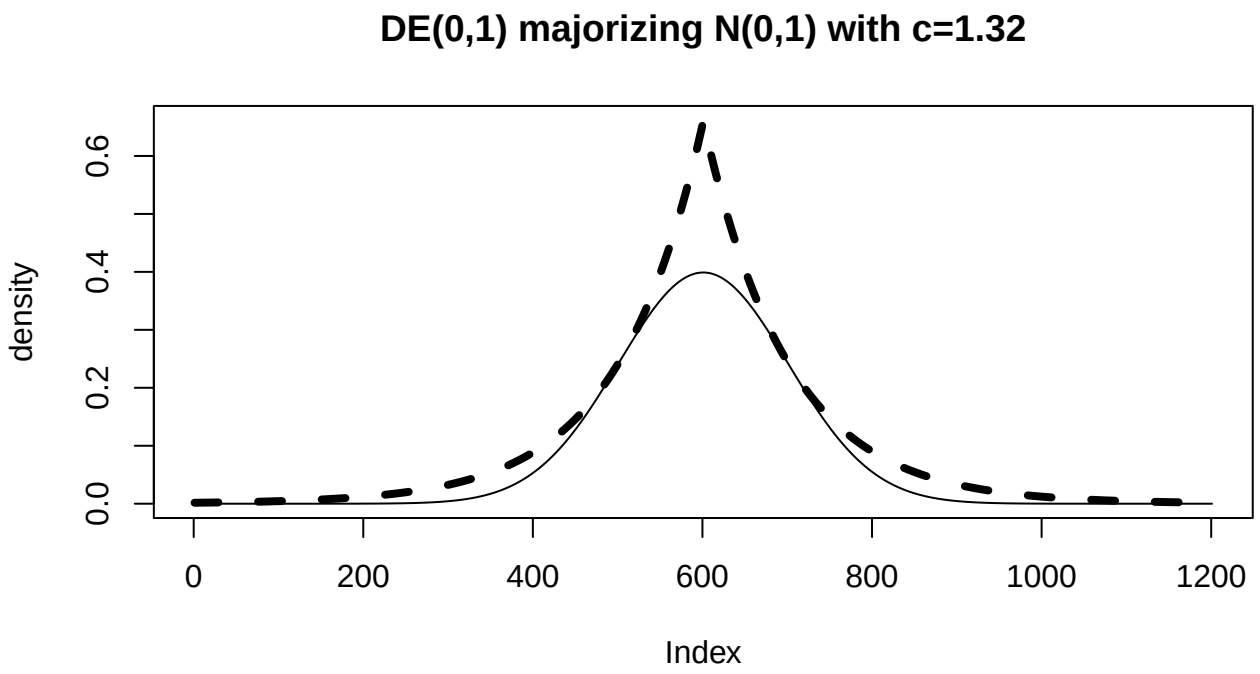


Acceptance/rejection method

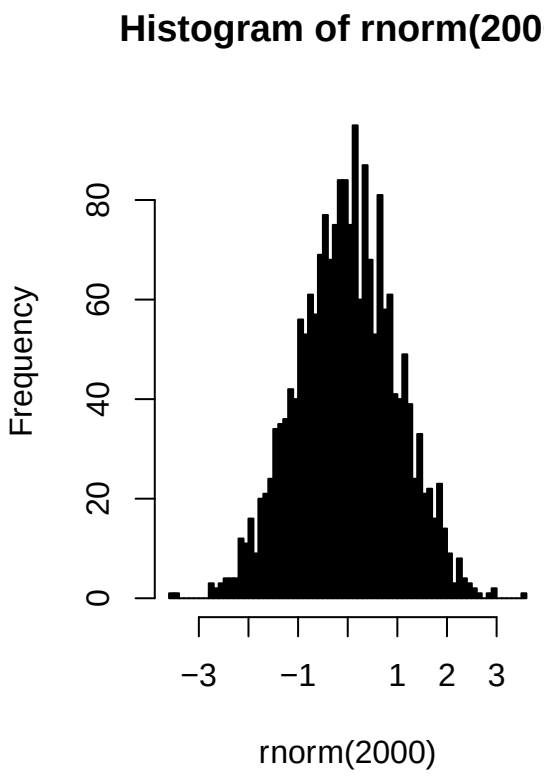
We choose c by a for-loop that checks if all values of $cf_y(Y) > f_x(X)$ for $c \in (0, 2)$ by 0.01 intervals. Whenever this is true it will stop and give us the number c. This gives the following:

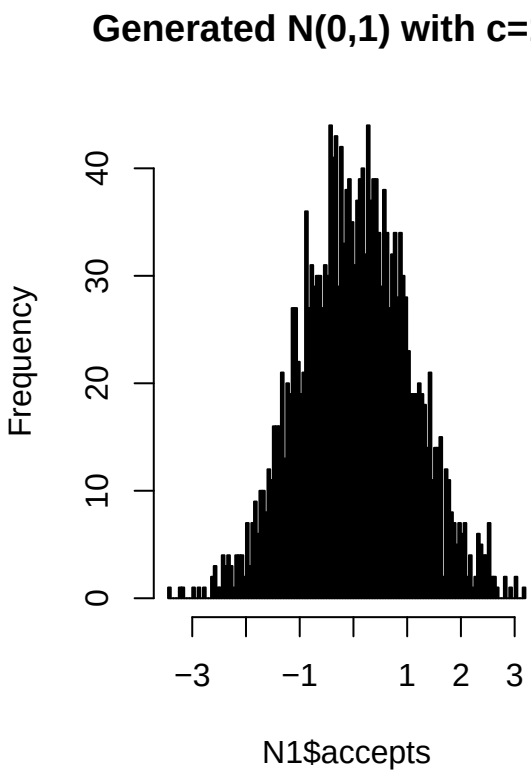
```
## Smallest c is 1.32 using a sequence (0,2) with 0.01 intervals.
```

The following shows the majorizing function $cf_y(Y)$ over $f_x(X)$,



Now we follow the acceptance/rejection-method step by step. First we generate a $Y \sim DE(0, 1)$ and a $U \sim U(0, 1)$. If $U \leq \frac{f_x(Y)}{cf_y(Y)}$ then we accept Y. We use the $dnorm(Y)$ function for $f_x(Y)$, and $ddoublex(Y)$ for $f_y(Y)$. We continue this process until we have accepted 2000 random numbers, and this set will hopefully be distributed as $N(0, 1)$. The following is a plot of 2000 numbers generated from rnorm() and the numbers generated by the acceptance/rejection-method.





We see that the generated numbers seem to be distributed as $N(0, 1)$.

The expected rejection rate is, $ER = 1 - 1/c = 0.242$ and we have observed,

```
round(N1$rej.rate,3)
```

```
## [1] 0.251
```

which is obviously close.

Code:

```
## ----echo=FALSE----------------------------------------------------------
library(knitr)
opts_chunk$set(echo=FALSE)

## ------------------------------------------------------------------------
setwd("/Users/niclaslovsjo/Library/Mobile Documents/com~apple~CloudDocs/Kurser/Comp stat/t3")

## ---- echo = FALSE,message=FALSE----------------------------------------
# 1.
library(xlsx)
pop <- read.xlsx("population.xls", sheetIndex = 1, startRow = 6,
                 colIndex = 1:7, encoding = "UTF-8")
pop <- subset(pop, pop$Code>30)

## ------------------------------------------------------------------------
# 2.
extract <- function(data = pop, n = 1){
  count <- 1
  cities <- c()
  while(count<=n){
    pop_sum <- sum(data$Population)
    data$cumulative_pop[1] <- data$Population[1]
    for(i in 2:nrow(data)){
      data$cumulative_pop[i] <- data$cumulative_pop[i-1]+data$Population[i]
    }
    data$ratio <- data$cumulative_pop/pop_sum
    val <- runif(1)
    cities[count] <- as.character(data$County.Municipality[which.min((data$ratio-val)<0)])
    data <- data[-which(data$County.Municipality==cities[count]),]
    count <- count+1
  }
  return(cities)
}

## ---- echo=FALSE--------------------------------------------------------
# 4.
cities <- extract(n = 20)

## ---- echo = FALSE------------------------------------------------------
size <- c()
for(i in 1:length(cities)){
  size[i] <- pop$Population[pop$County.Municipality==cities[i]]
}
plot(sort(size, decreasing = TRUE), type = "l", ylab = "Number of inhabitants")

## ---- echo = FALSE------------------------------------------------------
# 5.
breaks <- seq(0,850000,25000)
h1 <- hist(pop$Population, breaks = breaks, plot = FALSE)
h2 <- hist(size, breaks = breaks, plot = FALSE)
h1$density <- h1$counts/sum(h1$counts)
h2$density <- h2$counts/sum(h2$counts)
```

```r
plot(h1, col=rgb(0,0,1,1/4),freq = FALSE)
plot(h2, col=rgb(1,0,0,1/4), freq = FALSE, add = TRUE)
legend("top", c("All population", "Our sample"), pch = c(19,19), col = c(rgb(0,0,1,1/4), rgb(1,0,0,1/4))

## ----include=FALSE-----------------------------------------------------------
library(smoothmest)

## ----------------------------------------------------------------------------
U<-runif(10000)
U1<-runif(10000)
X<-c()
for (i in 1:10000){
  ifelse(U[i]<0.5,X[i]<-log(U1[i]),X[i]<--log(U1[i]))
}
X1<-c()
for (i in 1:10000){
  ifelse(U1[i]<0.5,X1[i]<-log(2*U1[i]),X1[i]<--log(2-2*U1[i]))
}

## ----fig.width = 8, fig.height = 3,fig.align='center'------------------------
par(mfrow=c(1,3))
plot(ddoublex(seq(-8,8,0.01)),type="l",main="DE(0,1)",ylab="density")
hist(X,breaks=1000,main="Method 1")
hist(X1,breaks=1000,main="Method 2")
par(mfrow=c(1,1))

## ----------------------------------------------------------------------------
lap<-ddoublex(seq(-8,8,0.01))
norms<-dnorm(seq(-8,8,0.01))
c<-0
for (i in seq(0,2,0.01)){
  if(all(i*lap>norms)){
    cat("Smallest c is",i,"using a sequence (0,2) with 0.01 intervals.")
    c<-i
    break
  }
}

## ----fig.width = 7, fig.height = 4,fig.align='center'------------------------
plot(1.32*ddoublex(seq(-6,6,0.01)),type="l",main="DE(0,1) majorizing N(0,1) with c=1.32",ylab="density"
points(dnorm(seq(-6,6,0.01)),type="l")

## ----------------------------------------------------------------------------
gen_fun<-function(c){
  accepts<-c()
  runs<-0
  non_accepts<-c()
  while(length(accepts)<2000){
    y<-rdoublex(1)
    U<-runif(1)
    X<-dnorm(y)
    Y<-ddoublex(y)
    ifelse(U<=X/(c*Y),accepts<-c(accepts,y),non_accepts<-c(non_accepts,y))
```

```
    runs<-runs+1
  }
  return(list(accepts=accepts,non_accepts=non_accepts,rej.rate=1-2000/runs))
}
N1<-gen_fun(c)

## ------------------------------------------------------------------------
par(mfrow=c(1,2))
hist(rnorm(2000),breaks=100,col="black")
hist(N1$accepts,breaks=100,col="black",main="Generated N(0,1) with c=1.32")

## ----echo=TRUE-----------------------------------------------------------
round(N1$rej.rate,3)

## ----code=readLines(knitr::purl('/Users/niclaslovsjo/Library/Mobile Documents/com~apple~CloudDocs/Kurs
## NA
```